Remote Agent Management

Microservers. A case study.
(draft, ver. 0.16)

Computer Sciences Department
Purdue University, February 2000

Radu Sion *(sion@cs.purdue.edu)*
Ladislau Bölöni *(boloni@cs.purdue.edu)*

Keywords: multi-agent systems, distributed management, microservers, object browsing

The Microserver Online
(*http://www.cs.purdue.edu/homes/sion/micros*)

Contents

1. Introduction …………………………………………………………………………… 1
2. Remote Web-Based Access for Multi-Agent Systems……………………………… 2
   2.1. Motivation. Issues………………………………………………………………….. 2
   2.2. Microserver. The Solution. …………………………………………………….. 3
   2.3. Access Points. Security. Examples…………………………………………… 3
3. Usage. Microserver Deployment. ……………………………………………………5
   3.1. Scenarios. ……………………………………………………………………….. 6
   3.2. Case study: Direct Web Access to Objects. …………………………………. 7
   3.3. Working report: Agent Monitoring in BOND. Web Management ………….. 8
4. The Context. Conclusions. Further research …………………………………………. 10
5. References ……………………………………………………………………………. 12
6. Appendix.

Abstract

This paper presents an approach for monitoring and control of mobile agents and active objects in a networked environment where objects and associated properties are highly dynamic and thus hard to control, maintain and debug. We propose Web-based access to collections of active objects and introduce the concepts of *accesspoint* and *microserver*. An accesspoint is a handle to a collection of objects, visible from the outside world. A *microserver* is a ultra-light software thread associated with an access point and a corresponding collection of objects. The microserver is accessible through external protocols such as HTTP or SOAP [5]. A microserver enables dynamic access to the system's active objects including agents in a transparent and portable fashion. A microserver can be deployed at any access point in the system, in a transparent manner, without interferring with the normal system usage. We present the design and implementation of microservers in the frame of a multi-agent system. As a case study we discuss the integration of a microserver in the Bond Agent Framework.

Note: This Paper was accepted and is to appear at the 2000 Congress on Evolutionary Computation, La Jolla Marriott, San Diego, July 2000 (Intelligent Agents Session)

1. Introduction

One of the challenges of multi-agent systems [1] [8] [9] [10] is to control and monitor agents in a highly dynamic environment. This raises several other problems such as consistency preservation, overhead minimization as well as the ability to offer unified access interfaces to different system components and agents.

Related main issues include integration with third party developments and alternative platforms, the degree of system openness (deriving partly from the accessability of the composing entities), the lack of standardization.

From the end-users point of view, some existing frameworks [12], [13] present custom implementations (ex. object editors, online management etc). However from the developer's point of view, integrating an existing system with third party components as well as issues like overall system monitoring and debugging should be made more easy and should require minimal system changes from the nominal (i.e. without debugging, monitoring etc.) usage.

We believe that current technologies and frameworks, enabling portability and global distributivity as well as increasing user-demands for mobility, unified access and ease of use, bring about the need for portable, transparent integration of existing resources through a unified interface.

We propose the ***microserver*** concept to enable dynamic access to the system's running agents and their properties. A *microserver* is an ultra-light software thread managing predefined *access-points* in the system. The accesspoints enable external access through an appropriate external common-use protocol (ex. HTTP) to the entities (e.g. agent's properties) related to the corresponding access points.

The paper is structured as follows. Section 2 presents the motivation and concepts of a microserver and accesspoint. Section 3 introduces main usage scenarios in deployment of microservers, as well as our hands-on experience in implementing an HTTP version of the concept. Section 4 presents related work and further research.

2. Remote Web-Based Access for Multi-Agent Systems. The Microserver.

Recent years brought about major transformations and split of concepts related to the term "agent". The increasing globality of networks as well as the emergence of related killer-applications like the Web in the case of the Internet, led to the apearance of a new entity called "agent" with certain characteristics, (e.g. autonomy, agency etc.), mainly *in the frame of a networked environment*.

In the following we present several related problems needing consistent and simple solutions, which led to the microserver concept.

2.1. Motivation. Issues.

One of the important issues especially related with network agents is the "remote" (in terms of network locality) agent access and control. "Access" refers mainly to the ability to *read* from within the agent's local data model, enabling monitoring and debugging. "Control" denotes the ability to trigger state transitions in the agent by changes in it's data model and remote calls to the agent or to a corresponding agent control authority.

Also, the ability to consistently implement global state aquiring protocols (e.g. snapshots and events [6]) in the frame of a heterogeneous system requires a portable protocol and access pattern, supported by all the participants.

The *microserver* enables dynamic access to the system's composing entities in a transparent and portable fashion. It's main functionality relies on the ability to easily deploy it at any *access point* in the system, involving a minimal interference with the rest of the system's composing entities.

Note: Although the concepts and interfaces introduced here are valid in any distributed platform we base our presentation on systems built mainly around the Java and Internet (TCP/IP) technology. The reasoning behind this decision lies with the de facto state of existing implementations. Also we feel that current Java technology offers just enough functionality and needed amount of concepts to develop sound, cross-platform implementations without sacrificing any desired functionality. We plan to write implementations also for other platforms and languages (e.g. C unix).

2.2. The Microserver. The Solution.

The *microserver* is an light software thread started at given *access-points* (see bellow) defined inside the system. It enables access through an appropriate protocol (such as HTTP or SOAP [5]) to the agent's properties related to the corresponding access point.

The notion of *lightness* in this context refers to the amount of resources used by the operating microserver inside the overall system. In an intuitive manner we define a piece of running code as being *light* if the behaviour of the running environment (in this case the actual accesspoint and the entire system within which it runs) is not changed signifiantly by the activities performed by the microserver code, in any manner. In reality this of course is an approximation but helps understanding many important design decisions (ex. maintaining a single self-sufficient microserver module).

Note: With respect to the concept of *lightness* it may be interesting to note that Heisenberg's rule simply cannot be beaten, although it's always challenging and fun to try. Software systems might be just enough a frame to assure provisions for decent aproximations. In our case, the microserver has to be offered access to IP Sockets, has to be able to allocate memory for certain (admittedly small) datastructures/objects. Because of a multithreaded design, it also has to be able to create and start threads featuring copies of itself.

The microserver is *"in charge"* of its corresponding access-point and *serves/exports* control primitives. In the case of the *property access point* (see bellow) this translates to the ability to remotely get() and set() property values within the access point.

By deploying microservers, many of the issues outlined previously can be solved. Agents can be controlled by remote calls and access to the exported agent model data. Agents can be monitored by being able to read/inquire agent properties at runtime. The usage of an external protocol such as HTTP, with it's wide acceptance, being a de facto standard, assures a high degree of openness and portability.

2.3. Access Points. Security.

Access points allow access to the system's internals, subject to the *selectivity* of the access point. One of the main features of an accesspoint is it's ability to implement and deploy various security schemes. The selectivity of the accesspoint is defined as being the extent in which it uncovers the properties to the caller.

An important access point is the ***property access point*** which basically allows public access to the agent properties. An example drawn out of one of the main usage scenarios (distributed monitoring) would be the language's public name space from within the running Java Virtual Machine (or, for that matter, within any other running system), name space composed of variable properties which may be ***.set()*** and ***.get()***.

Another challenging and important case is the ***method invokation access point*** which aims at enabling RMI alike calls to the agent (or to objects) itself. This presents a number of interesting issues to cope with, for example the format and serialization of corresponding call arguments and result.

In the case of both accesspoints described, the microserver plays the role of a simple ***access protocol translator***, from the internal object messaging protocol (inside the Java Virtual Machine) to the external (e.g. HTTP) access protocol. An analogy that can be made is with the RPC [14] and RMI [15] mechanisms. If the access point includes *callable* entities (i.e. methods) the access patterns are similar to remote procedure calls.

Directory access point. Directory Microserver.

Several access points can be organized to form hierarchical configurations. This enables unified access to all of the system's properties. The implementation of this scheme is done through d*irectory accesspoints*, managed by a *directory microserver*.

Another aproach would be to directly deploy accesspoints (as opposed to microservers) in a hierarchical structure. The decision on which solution to use is linked to properties of the target system like mobility, scale, computing power, feasability of hierarchical data access patterns to composing agents and objects etc.

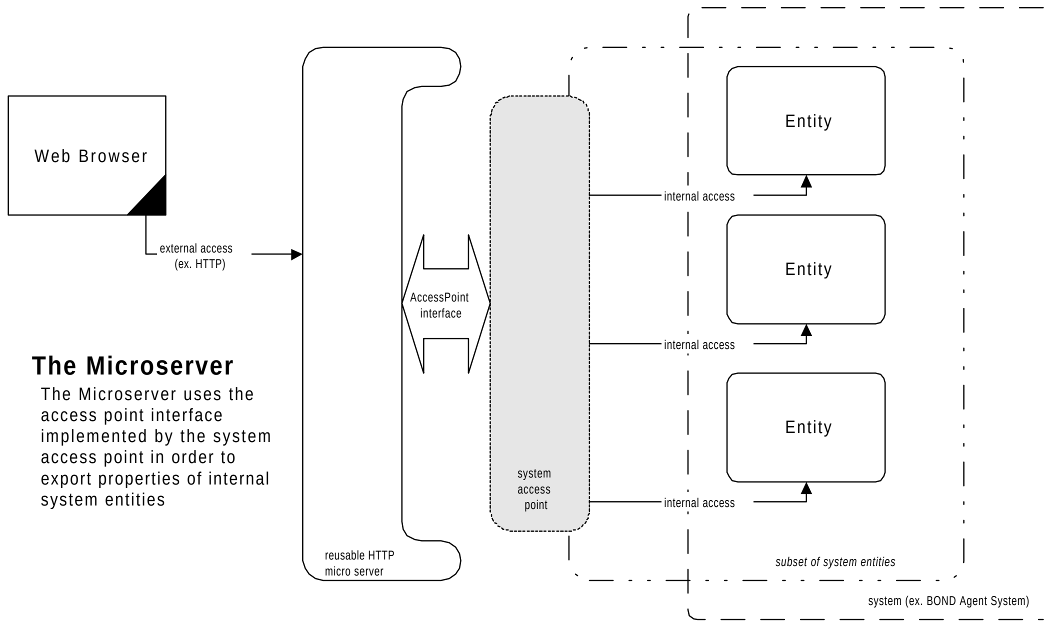
Internal access point

A microserver approach as presented above enables external access to internal system properties. But the concept is not limited to this type of usage. In fact, the microserver as such may be deployed inside a system as to support actual internal property access requirements. An *internal accesspoint* is an abstraction defining an accesspoint that is of internal use and that prohibits external access.

Examples

Currently we implemented several general use accesspoints, reusable to a great extent in any agent system built on top of Java technology. We also implemented more specialized accesspoints, pertaining to the Bond Agent Framework.

Directly reusable access points include the *java object field accesspoint* and the *java object method invocation* accesspoint. The Java Object Reflection Field Accesspoint exports and allows access to any Java Object's property in a transparent manner (from the Object's perspective). It can be used to remotely monitor changes in the Object's state and data. The Java Method Invocation Accesspoint exports callability of methods on objects. This implies the ability to serialize call arguments as well as getting back method return. Issues such as protocol timeouts, stream serialization issues are particulary challenging.


Web Browser
external access (ex. HTTP)
The Microserver
The Microserver uses the access point interface implemented by the system access point in order to export properties of internal system entities
AccessPoint interface
reusable HTTP micro server
system access point
internal access
internal access
internal access
Entity
Entity
Entity
subset of system entities
system (ex. BOND Agent System)

Both accesspoints discover object properties, fields and methods (i.e. without involving the Object's code itself) by use of the java reflection mechanisms. In this respect the accesspoints can be reused as "observers" within any system.

In the case of Bond we implemented accesspoints corresponding to the Bond Object and the Bond Directory. This enables arbitrary calls to any bond Object’s method, access to all dynamic and static properties, access to all directory registered entities.

By deploying these access points we enable external calls to the agent factory for example allowing agent creation and control. Running agent’s models and strategies are also accessible, making thus signifiant steps towards distributed Web-based debugging.

3. Usage. Microserver Deployment.

This chapter is dedicated to actual usage scenarios and real world deployment of microservers. We introduce a set of envisioned usage scenarios. We also present two case studies of our experience in deploying microservers. Direct web access to a simple object is the most simple and straightforward application of the microserver. We also elaborate on current work on integrating the microserver into the Bond Agent System.

3.1. Scenarios

Scenarios involving microserver deployment include the ability to easily export local objects to external applications, to browse/access entire hierarchies of objects, remote runtime debugging and monitoring as well as online system administration and entity control.

All of these usage scenarios have seen several solutions and different implementations. Our approach is to allow the creation of an unique, consistent interface assuring also portability and compatibility with existing systems. The usage scenarios presented in this section illustrate this.

Portable Data Retrieval.

Transfering data objects in a transparent and portable fashion constitutes a challenge in many existing OO systems. Specialized mechanisms such as the Java Object Serialization provide custom solutions. Those solutions always imply system awareness and compatibility at the receiver side.

Note: The Java Serialization Mechanism for example presents many problems and unsolved issues (e.g the receiver side has to know the class type and version of the transferred Object etc.) constituting serious drawbacks in communicating with other platforms.

Being able to export objects identified by Uniform Resource Locators (URL) through deployment of HTTP microservers at the originator’s side together with the ability to properly format the data (based on the client’s request) enables legacy platforms (e.g. Perl and C programs) to access data from within remote systems. (e.g. this can easily lead to deployment cases when for example a unix/windows perl script is used to control and monitor a remote agent’s behavior)

Remote Agent Management.

Obviously, Remote Agent Management is one useful and very appealing deployment case. The ability to call arbitrary methods as well as to set any fields corresponding to any entity within the system, delivers virtually total control over the running system. Harnessing this control may imply designing appropriate web entry point documents containing references to various access points (controlled by microservers) within the system.

Distributed Debugging and Tracing.

Traditional code debugging and runtime tracing is usually done by using specialized tools and debugging environments. Multi-agent systems composed of highly-dynamic mobile agents and active objects pose new challenges related to the inability to access remote entities in a unified way. The microserver constitutes a solution.

Monitoring.

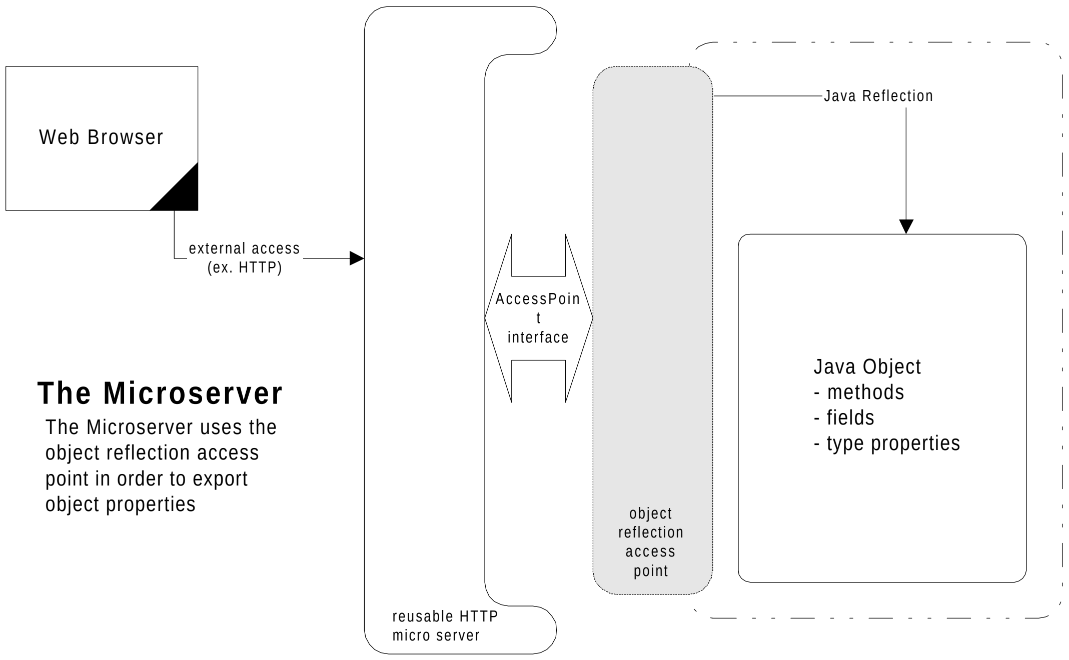
Another usage scenario closely related to distributed tracing is distributed monitoring. Integration with traditional tools such as Perl produce powerful solutions.

Deployment example: actual monitoring takes place inside a simple Perl script which accesses the system through an active microserver. If for example critical conditions arise, the perl script can take the decision to change the overall state of the system, to enlarge the available disk space or even to send an email or pager notice to the system administrator.

Note: As outlined before, several envisioned usage scenarios have emerged. Many others exist. Any scenario requiring remote access to a system's internal objects in a unified an portable manner can be solved by microserver deployment.

3.2. Case study: Direct Web Access to Objects.

In this section we present the most simple yet powerful case of deployment of microserver, the direct web access to objects. In this case, a given java object (e.g. a component within a more complex system) is exported by a microserver through the object reflection access point outlined previously.


Web Browser
external access (ex. HTTP)
The Microserver
The Microserver uses the object reflection access point in order to export object properties
AccessPoint interface
reusable HTTP micro server
object reflection access point
Java Reflection
Java Object
- methods
- fields
- type properties

As can be seen form the figure, the accesspoint actually *wraps* an associated Object and uses internal access mechanisms (i.e. method calls). The example java source code associated with the above figure would be:

```
[…]

// arbitrary object:
Object obj;

[…]

// creating access point for arbitrary object
AccessPoint objacc = (AccessPoint)(new ObjectReflectionNameSpace(obj));

// starting new microserver on port 7099,
// exporting the entire object reflection namespace
// with error output to the standard error output stream
(new HTTPMicroServer(7099, objacc, System.err)).run();

[…]
```

Two steps are illustrated in the code above. First, an access point corresponding to the object to be exported is created. The object reflection name space is an accesspoint that uses the java reflection mechanisms in order to transparently export all of the object's properties. Next, the reusable http microserver in charge of the newly created accesspoint is created and started.

After starting the microserver, the object is accessible externally (ex. through a web browser) by using an URL of the form:

http://smart.cs.purdue.edu:7099?name=a&action=get&format=html

If the object contains a field named "a", the URL above accesses that field and outputs it formatted accordingly to a HTML formatting convention. See the appendix for more documentation on our Java HTTP microserver implementation and parameters.

3.3. Working report: Agent Monitoring in BOND. Web Management.

Bond is a Java based distributed object system and agent framework, with an emphasis on flexibility and performance. It is composed of a *core level*, containing the object model and message oriented middle-ware, a *service layer* containing distributed services like directory and persistent storage services and the *agent framework*, providing the basic tools for creating autonomous network agents. The agent framework also includes a database of commonly used strategies which allow developers to assemble agents with no or minimal amount of programming.

BOND. Basic Concepts

Bond consists of several packages including Bond core and Bond agents. Brief descriptions of relevant parts and supported functions follows.

Bond Core

The Bond Core package supports basic object manipulation, inter-object communication, local directory and local configuration services, a distributed awareness mechanisms, probes for security and monitoring functions, and graphics user interfaces and utilities.

A *Bond Object* is a collection of data fields and methods. The data component of an object consists of several fields and dynamic properties. Each object is identified by a unique bondId. The granularity of Bond objects ranges from simple objects, e.g. messages, to complex objects like a directory server. Some objects are passive, e.g. a message or a meta-object associated with a data file, others are active they have one or more threads of control. Active as well as passive objects can communicate using the say() method.

A *Bond Resident* is an active container consisting of several threads of control, a local directory and a set of objects. It is the main runtime environment for bond Agents and general Objects. A *shadow* object is a proxy for a remote object.

Bond Agent

The Bond Agent package supports dynamic assembly and control of software agents. The package consists of an agent framework based upon the Bond agent model, a language for agent definition, an agent factory probe, strategy databases, and graphical user interfaces. A Bond *agent* is a collection of state machines sharing a model of the world and an agenda. Every state of the state machine has an associated strategy. The *Blueprint* agent definition language is an interpreted language, which allows dynamic assembly of agents from components. It allows specification of strategies via embedded languages.

Communication

Bond supports several different communication mechanisms, such as reliable message delivery, best-effort message delivery and multicast message delivery. Each object in Bond has the ability to send and receive messages using the bondCommunicator class. To make bondCommunicator independent and hide the details of underlying communication mechanism, Bond uses an abstraction called *communication engine* to describe the interface between bondCommunicator and the real communication objects.

For more information on the Bond Agent System please access ***http://bond.cs.purdue.edu*** or email us at ***bond-devel@cs.purdue.edu***.

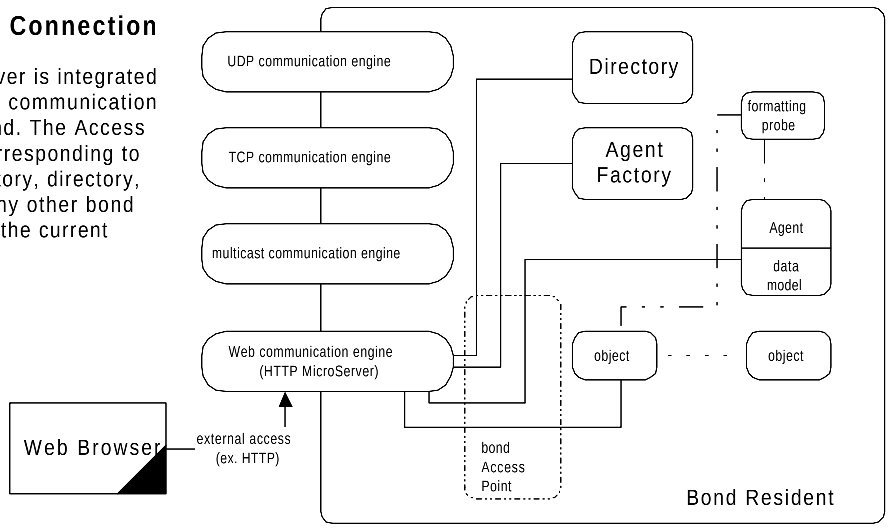
HTTPMicroServer

In Bond we implemented access points corresponding to the Bond Object and the Bond Directory. This enables arbitrary calls to any Bond Object method, access to all dynamic and static properties as well as access to all directory-registered entities, including active agents.

We integrated the HTTP Microserver implementation as another communication engine in Bond. This approach is consistent with the overall Bond architecture. It allows direct browser access to any bond component including the agent factory, as well as the ability to integrate the Bond framework with other systems offering the same type of external access (ex. by using a Microserver and a general purpose access point).

BOND Web Connection

The Microserver is integrated naturally as a communication engine in Bond. The Access points are corresponding to the agent factory, directory, agents and any other bond Object within the current Resident



Remote agent creation and migration control is much simplified by the availability of direct access to the agent factory. By accessing simple URL's calls inside the system can be triggered.

Web Integration

Current work focuses mainly on web interface issues such as HTML formatting and transparent access point hierarchization. This is needed in order to fully benefit from the facility provided by the running microserver and the corresponding deployed accesspoints.

4. The Context. Further research. Conclusions.

Recently Microsoft announced an interesting proposal to IETF called Simple Object Access Protocol (SOAP) [5] proposal that may take the microserver concept even further. The proposal defines a new protocol for remotely accessing objects. The protocol is based on an extension of the base HTTP protocol and defines a minimal set of conventions for invoking code using XML and HTTP.

We started the implementation of a SOAP compliant microserver. Research is still needed, especially in specifying exact serialization formats and semantics.

Another interesting idea would be the deployment of server push related technology (including persistent connections) to make subscription based monitoring possible. In this frame, the microserver has to support persistent HTTP connections. Once a connection is established to the microserver, it can be used to notify the client of occuring changes in the monitored access point's entities. Issues such a protocol timeouts, stream serialization issues as well as feazability of integrating persistent connection support in the microserver need to be subject to careful design.

With respect to the Bond implementation, another interesting issue to explore would be the deployment of probes in accessing the bond object. Probes are objects attached dynamically to Bond objects to augment their ability to understand new subprotocols and support new functionality. In this case, either a direct microserver based probe can be used, enabling the object to directly export it's own functionality or specialized probes may be designed in order to enable the system's web communication engine (microserver) to access the object. This is subject to further design changes.

One case of interest is the the case of migrating agents. In this case attaching a microserver to the running agent enables persistent monitoring of the agent. Issues arise in maintaining contact to the agent after the migration process completes and the agent is now running on another computer with a different IP address. One solution we envision and plan on implementing is to issue an redirect directive to the client (either using Javascript or HTML META tags) just before the migration starts. The redirect specifies a set of new locations where the microserver may be running on the new machine.

This paper presented an approach for monitoring and control of mobile agents and active objects in an environment where objects and associated properties are highly dynamic and thus hard to control, maintain and debug.

We proposed external access to collections of active objects and introduce the concepts of *accesspoint* and *microserver*. An accesspoint is a handle to a collection of objects, visible from the outside world. A *microserver* is a ultra-light software thread associated with an access point and a corresponding collection of objects.

The microserver can be accessible through various external protocols and extensions of known protocols such as HTTP or SOAP [5]. A microserver is deployed at any access point in the system, in a transparent manner. This paper focused on the design and implementation of microservers in the frame of a multi-agent system. As a case study we discussed the integration of a microserver in the Bond Agent Framework.

5. References

[1] Bond online, *http://bond.cs.purdue.edu. Email: bond-devel@cs.purdue.edu.*

[2] A gentle introduction to Bond, *http://bond.cs.purdue.edu/guide/Intro.ps*

[3] Java Object Serialization, *http://java.sun.com/products/jdk/1.2/docs/guide/serialization*.

[4] The microServer online, *http://www.cs.purdue.edu/homes/sion/micros*.

[5] G. Kakivaya, A. Layman, S. Thatte, Microsoft Corporation, D. Winer, Userland Software, SOAP: Simple Object Access Protocol, IETF Informational Drafts, November 1999

[6] Ozalp Babaouglu, Keith Marzullo, "Consistent Global States", Distributed Systems Second Edition, ACM Press, 1993

[7] The W3C Resource Description Framework, *http://www.w3.org/RDF*.

[8] MIT's HIVE, *http://hive.www.media.mit.edu/projects/hive*.

[9] IBM Aglets, *http://www.trl.ibm.co.jp/aglets*.

[10] The Grasshopper Agent Platform, *http://www.grasshopper.de*.

[11] Radu Sion, Jet Virtual Medium, *http://www.cs.purdue.edu/homes/sion/jvm*.

[12] Jini, Sun Microsystems, *http://www.sun.com/jini*.

[13] E-speak, *http://www.e-speak.net*.

[14] Java RMI, *http://java.sun.com/products/jdk/rmi*.

[15] Sun Microsystems, Inc., Network Working Group, June, 1988, Request For Comments: 1057 RPC: Remote Procedure Call, Protocol Specification Version 2